ASSESSMENT AND TESTING OF INDUSTRIAL DEVICES ROBUSTNESS AGAINST CYBER SECURITY ATTACKS

F. Tilaro, B. Copy, CERN, Geneva, Switzerland

Abstract

CERN (European Organization for Nuclear Research),like any organization, needs to achieve the conflicting objectives of connecting its operational network to Internet while at the same time keeping its industrial control systems secure from external and internal cyber attacks. With this in mind, the ISA-99 [1] international cyber security standard has been adopted at CERN as a reference model to define a set of guidelines and security robustness criteria applicable to any network device. Devices robustness represents a key link in the defense-in-depth concept as some attacks will inevitably penetrate security boundaries and thus require further protection measures. When assessing the cyber security robustness of devices we have singled out control system-relevant attack patterns derived from the well-known CAPEC [2] classification. Once a vulnerability is identified, it needs to be documented, prioritized and reproduced at will in a dedicated test environment for debugging purposes. CERN - in collaboration with SIEMENS - has designed and implemented a dedicated working environment, the Test-bench for Robustness of Industrial Equipments [3] ("TRoIE"). Such tests attempt to detect possible anomalies by exploiting corrupt communication channels and manipulating the normal behavior of the communication protocols, in the same way as a cyber attacker would proceed. This document provides an inventory of security guidelines [4] relevant to the CERN industrial environment and describes how we have automated the collection and classification of identified vulnerabilities into a test-bench.

INTRODUCTION

The Internet has become essential for any organization or company that would like to conduct any sort of business. On one hand, in Industry this results in an improvement of the communication from the fabric floor network to the business level one; but on the other hand, the Internet has brought a lot of security problems which cannot be ignored because of the resulting damages and disasters. Thus, it is necessary to identify methods and procedures for achieving the dual mission of both exploiting the advantages provided by the Internet and at the same time keeping their industrial systems secure from external and internal attacks.

As we know, the number of hostile applications, worms and viruses is continuously increasing and hackers are more and more interested at exploiting common industrial control systems vulnerabilities [5]. As the well-known industrial control security expert Joe Weiss testified to the US Congress, in 2010 more than 180 security incidents were reported; it is worth to mention the Deepwater Horizon (BP Mexican Gulf oil spill) counting 11 casualties or the San Bruno gas pipeline explosion, with 8 casualties.

It is evident that industrial security cannot rely only on IT security techniques and properly configured network architectures (like firewall, network segmentation, antivirus software, access management, etc…) which, however, can provide only a partial protection for the entire system; this is easily explained if we consider the fundamental differences in the deployed hardware but also in the concepts of availability and manageability between IT and Industrial Systems. On the contrary, new methodologies aimed at securing industrial systems must be integrated into IT security standard practises.

This paper mainly focuses on testing industrial Ethernet-connection-based devices without going into details on network security configurations.

COMMUNICATION ROBUSTNESS

Any communication between two or more hosts is regulated by a protocol whose specification defines the formats, syntax and semantic rules for exchanging messages – called Protocol Data Units (PDUs) [6] - over a network. Protocol specifications are typically written in natural - not deterministic - language and sometimes cannot state how to handle all the possible faulty inputs; so many decisions are left to the implementer. Hence it is clear that distinct implementations of the same protocol handle PDUs in different ways.

Due to this heterogeneity, each implementation could present many issues related to services availability, performance and even security: once a hacker has figured out some malformed PDUs or sequences of them which are not properly handled, the entire system is under risk [4]. Ideally all these defects affecting the protocol implementations should be detected and fixed by the manufacturers, but they are sometimes actually detected and reported by external communities. This explains the need of performing automated testing of protocol implementations, especially for critical systems.

In this paper we present a methodology for automated testing of protocol implementation robustness, which must be seen as the ability of the system to handle exceptionally malformed PDUs and stressful network conditions, while maintaining the normal operational behaviour. This evaluation proves a greater ability of analyzed networked systems to survive in the face of

malformed input due to possible involuntary mistakes or explicit attack attempts.

As explained in the following section of the document, the enumeration of all possible faulty PDUs for each protocol is exponential in the number of protocol fields; so it is necessary to devise a strategy to reduce the number of possible malformed PDUs to generate, while still detecting security issues accurately. This can be achieved by exploiting the knowledge of communication experts to distinguish the test cases which could be really interesting, from the ones that are redundant and maybe even duplicated. The approach we are proposing is used to generate individual malformed PDUs or sequences of them in a systematic manner according to a specific grammar definition. Grammars define a set of syntactic and semantic rules to cover a specific domain of tests (generally related to a specific communicational protocol header); if the protocol implementation cannot handle invalid packets correctly, anomalous behaviour may occur and needs to be detected.

The approach, we are proposing, must be generic enough to be applied to all protocols even to industrial ones which exhibit really specific properties and features.

ANALYZED TOOLS AND SECURITY STANDARDS

In the recent years the increasing number of cyber security related incidents affecting industrial control systems has made vendors of critical infrastructure to pay more attention to security issues. Unfortunately, there are no complete and comprehensive standards whose specifications can be followed to protect any critical system; nonetheless several initiatives have been started with the objective of improving the security level and the robustness of industrial systems.

The ISA Secure Program [7], on the basis of ISA 99 Standards specifications, has produced a certification program with three levels of recognition for a device security assurance.

Among the commercial products - that we have already tested and used in our initial testing phase - Wurldtech Achilles [8] must be mentioned; it is an all-in-one vulnerability scanner specifically designed for suppliers of industrial process automation, control and safety systems such as: Programmable Logic Controllers, Supervisory Control and Data Acquisition (SCADA) Devices, Distributed Control Systems and Critical Systems. Wurldtech has also developed a proprietary certification with the purpose of evaluating and assuring the robustness and resilience of industrial products. However, to overcome the Achilles platform's proprietary aspects and limitations in terms of supported network protocols and attack techniques customization support, we have designed and implemented the TRoIE test-bench [4]. Moreover Wurldtech Inc. is also providing a valuable contribution to the preparation of industrial cyber security standards, such as ISA99.

In the wide range of open-source security frameworks and tools we have integrated into our test-bench the Open Vulnerability Assessment System [9] (OpenVAS); born as a branch of the popular Nessus2, OpenVAS provides a flexible environment for the development and deployment of new vulnerability scanning techniques. So far this framework does not offer a real wide range of tests developed specifically for industrial and process control environments. This field has been attracting more and more interest in the last few years: for instance, the Nessus vulnerability scanner has recently introduced an internal package finalized to industrial protocols and systems tests.

ENHANCED PROTOCOL FUZZING TESTS WITH GRAMMAR DEFINITION

When assessing the cyber security robustness of devices, *attack patterns* [10] can be used to categorize the discovered vulnerabilities. An attack pattern is expressed as "a series of repeatable steps simulating an attack against a system".

Such classification is useful to identify the cause of vulnerability and the potentially related well-known solution. In our experience we focused on Fuzzing and Grammar tests because of their effectiveness at probing devices for security vulnerabilities.

Protocol Fuzzing is a very effective testing technique generally deployed to generate valid and invalid packets with "randomized" header field values; its main purpose is analyzing the behaviour of a specific protocol implementation or functions of the protocol stack by injecting unexpectedly malformed input parameters values.

Fuzzing, according to the first, narrower definition, might be characterized as a blind fishing expedition that hopes to uncover completely unsuspected problems in the software under test. However it must be also admitted that for many interfaces, the idea of simply injecting random bits works poorly generally because of the wide domain of test. For example, injecting a web interface with randomly generated strings will have the only effect to detect only invalid URLs: they will be mainly rejected suddenly, perhaps by a simple parser - acting a sort of protection - checking for valid URLs. This is why completely random fuzzing is a comparatively ineffective way to uncover problems in a generic system. On the contrary Fuzzing acquires more efficiency when it is combined with "intelligent" techniques. Microsoft refers to this as "smart fuzzing" [11]: it is not a random testing anymore, but the generation of the tests is led by the target specifications; hence an initial knowledge of the system under test is required. In our case the fuzzing system is fed with some grammar rules which specify the part of the protocol headers to fuzz and the strategy we want to follow for the generation of the injected packets. One of the most important benefits coming from this kind of testing strategy is the ability to systematically and

predictably explore the input space, instead of having to rely on randomly generated noise.

Our final strategy is a mix of fuzzing and syntax testing: syntax is used to translate the knowledge of security experts into rules, which determine the packets generation and injection. In any case, the creation of effective syntax tests requires a deep understanding of the multiple protocols under test and their possibly stateful interconnections.

A PROTOCOL FUZZING SYSTEM IMPLEMENTATION

In the initial phase of the project, we analyzed a wide range of available specialized fuzzing utilities. Some of them exhaustively iterate through a designated protocol, whose specification are known in advance; so they could be used to stress test a variety of applications that support that protocol. We then opted for other generic fuzzers capable of fuzzing arbitrary protocols and file formats by performing simple, in principle non-protocol-aware, data mutations such as bit flipping or byte transposing. Although these fuzzers are effective against a wide range of common applications, we often have a need for more customization and thorough fuzzing for proprietary and previously untested protocols (we remind that our main targets are industrial control devices with proprietary specifications). This is where fuzzing frameworks become extremely useful. Among them we explored three open source fuzzing frameworks: SPIKE, Sulley Fuzzing Framework and Peach. Eventually the last framework has been selected for its flexibility and simplicity at developing specific customized protocols fuzzing tests. Nonetheless Peach provides some utility to convert Wireshark captured network traffic file into its protocol model format. It also includes several modules for target faults detections; but unfortunately they cannot be used in our scenarios where a custom monitoring system has been developed to observe the behaviour of the industrial devices under tests.

Peach Fuzzing framework [12] is an open source cross-platform fuzzing framework written in Python. It provides with a well designed software components, like mutators, transformers, protocols definitions, publishers, groups, etc... These components can be extended and chained together to simplify the generation of customized complex data types. As underlined before, Peach offers a very high object oriented abstraction level through the definition of pure python classes: it allows a tester to focus on the individual subcomponents of a given protocol, later tying them together to create a complete fuzzer scheme. As a result this approach generously promotes the code reuse for the development of new fuzzers. We have customized this framework by defining:

- new publishers tailored at injecting the generated data into the specific protocol formats we want to test;
- new mutators responsible for the generation of data ranging values and types;
- new transformers to encode the data values in a proper way;
- new mutation strategies which establish the algorithm to follow at combining the protocol fields values;
- new agents and monitors to integrate the fuzzing system with the external monitoring one: in case of target failure detection the fuzzing test saves its current state to restart it later from the last point.

Once the framework has been customized through the implementation of the previously described components, we have started the creation of so called “PeachPit” files: they are XML files containing all of the information necessary to run a fuzzing test:

- the data model defines the structure, information type, and relationships in the data to be fuzzed;
- the state model: It additionally allows for the configuration of a fuzzing run including selecting a data transport (Publisher), logging interface, etc.
- Generic configuration to specify the publishers, agents and monitors to use, including their specific initial parameters values.

In line with the ISA-99 security standards specifications we have defined an independent testing and certification system for industrial control devices. The PeachPit file definition is not totally arbitrary, but aims at fulfilling the ISCI Communication Robustness Testing (CRT) requirements. In practises we have implemented within the Peach fuzzing framework a list of security tests classified by protocol and scope of test. However it must be said that this certification does not cover the industrial protocols yet; so, to overcome this limitation, we have defined our own tests by applying the same security concepts and guidelines used for the open protocols.

MONITORING

In any testing procedure it is essential for the tester to be able to determine if the behaviour of the system under test is symptomatic of vulnerability or anomalies either in the software or hardware.

This examination can be harder in security testing than in traditional functional testing, because the tester is not necessarily comparing actual program behaviour to expectations derived from specifications. Rather, the tester is often looking for unspecified symptoms indicating the presence of unsuspected vulnerabilities. Furthermore monitoring must be integrated into several automated procedures, which can be used to evaluate test outcomes and identify only the anomalous ones: this is especially true during high-volume test activities like fuzzing. Unfortunately there are no third-party test automation tools able to monitor generic industrial devices’ outputs and internal behaviour; so we have been forced to design and implement our own monitoring strategy. At least the “essential services” - following the ISCI CRT [6] naming convention - must be observed

during the testing activities. In our specific case the monitoring system should check and track any of the following effects: extension of the current PLCs scan cycle, excessive use of memory, CPU overuse, delays in the physical outputs and consistent delays in the periodic communication.

Once an anomaly has been detected, it becomes relevant to identify its cause, the specific packet or sequence of packets. Because of its complexity it is not generally easy to achieve this task: one anomaly could be the result of different environment variables, and the device's behaviour could not be affected by the single factors, but only by a specific combination of them.

ACHIEVEMENTS

The described strategy has already proven to be effective at detecting device robustness issues. Thanks to the performed testing analysis, it was possible to detect critical anomalies in the devices' software protocol stack implementations. These research findings have been directly reported to their proper industrial vendors in order to be patched and incorporated in subsequent firmware releases. These initial encouraging results have motivated the team to continue following and expanding this approach for the future of the collaboration between CERN and the automation industry.

CONCLUSIONS AND FUTURE WORK

The entire article is based on the assumption that any network protocol implementation is susceptible to accidental or malicious corrupted communication. For this reason, it is essential to perform robustness testing of these implementations for critical industrial systems. Our approach consists of analyzing protocol implementations by injecting malformed PDUs to corrupt the normal behaviour of the system. As a PDU typically has many fields, the number of possible syntactically faulty PDUs grows exponentially with the number of fields. In this document, we proposed a strategy to explore this huge test domain using a hybrid approach of fuzzing and syntax techniques, specifically developed to evaluate industrial device communication robustness. So far, not all the tests can be integrated into automatic tools, human analysis and management is necessary to discover and investigate specific possible failures.

Moreover it should be remembered that security analysis must be seen as a dynamic process which should be adapted according to new requirements, constraints and technological changes. So the testing techniques and methodologies defined in this document should be adapted and modified to fit incoming features and evaluate new functionality.

In the future, we will extend the scope of our analysis to the industrial supervision layer: the targets of our tests will not only be the individual devices but also SCADA systems in order to estimate the potential impact of malicious PDUs within the entire industrial network architecture.

REFERENCES

[1] ISA-99: Manufacturing and Control Systems Security, http://www.isa.org
[2] CAPEC (Common Attack Pattern Enumeration and Classification), http://capec.mitre.org
[3] F. Tilaro, "Test-bench for Robustness...", CERN, 2009
[4] B. Copy, F. Tilaro "Standards based measurable security for embedded devices", ICALEPCS 2009
[5] Stuxnet, http://www.langner.com/en/2011/08/04/stuxnet-and-beresford/
[6] A. Tanenbaum "Computer Networks", Prentice Hall 2000
[7] ISA Secure Program, http://www.isasecure.org/
[8] Achilles Satellite Security & Robustness Testing Platform, http://www.wurldtech.com/cyber-security-products/achilles-satellite/default.aspx
[9] Open Vulnerability Assessment System (OpenVAS), http://www.openvas.org/
[10] F. Tilaro, B. Copy "Guidelines for evaluating PLC security", CERN, 2010
[11] Howard, Michael & Lipner, Steve. The Security Development Lifecycle. Redmond, WA: Microsoft Press, 2006
[12] Peach Fuzzing Platform, http://peachfuzzer.com/